BAB II

LANDASAN TEORI

2.1 Diabetic Retinopathy

Jumlah orang yang didiagnosis menderita diabetes telah meningkat secara dramatis selama beberapa dekade terakhir, dan diabetes meningkatkan risiko berbagai penyakit mata, di mana *diabetic retinopathy* adalah salah satu yang paling parah (Xu, 2017). *Diabetic retinopathy*, juga dikenal sebagai penyakit mata diabetik, adalah suatu penyakit mata di mana pembuluh darah pada retina mengalami kerusakan sebagai efek samping atau komplikasi dari diabetes. Pada akhirnya, *diabetic retinopathy* pada tahap yang parah dapat menyebabkan kebutaan (Doshi, 2016). *Diabetic retinopathy* menjadi penyebab utama kebutaan pada populasi usia pertengahan (Xu, 2017). Adapun yang tergolong ke dalam populasi usia pertengahan adalah orang yang terdapat dalam rentang usia 45 – 60 tahun (Oxford, 2020).

Diabetic retinopathy merupakan proses progresif, sehingga para ahli medis menyarankan bahwa pasien penderita diabetes perlu melakukan *check-up* dan tes tidak kurang dari dua kali dalam jangka waktu setahun untuk mendiagnosis tanda-tanda awal dari penyakit yang mungkin muncul secara tepat waktu. Dalam diagnosis klinis saat ini, diagnosis dari *diabetic retinopathy* bergantung pada dokter mata yang memeriksa citra retina dan kemudian mengevaluasi kondisi pasien. Proses deteksi tersebut cukup sulit dan memakan waktu yang tidak sedikit sehingga mungkin menghasilkan banyak kesalahan. Selain itu, karena pasien diabetes jumlahnya sangat banyak dan terdapat kekurangan sumber daya medis di beberapa

daerah, banyak pasien dengan *diabetic retinopathy* tidak dapat didiagnosis dan diobati secara tepat waktu, sehingga berakibat pada kehilangan peluang untuk mendapatkan pengobatan yang baik dan pada akhirnya menyebabkan pasien tersebut kehilangan penglihatannya secara permanen (Wan, 2018).

2.2 Deep Learning

Deep learning (DL) adalah bagian dari *machine learning* yang didasarkan pada pembelajaran beberapa level representasi dengan membuat hierarki fitur, di mana level yang lebih tinggi didefinisikan dari level yang lebih rendah dan fitur level yang lebih rendah yang sama dapat membantu dalam mendefinisikan banyak fitur level yang lebih tinggi (Tharani, 2016). Struktur DL memperluas *Neural Network* (NN) tradisional dengan menambahkan lebih banyak *hidden layers* ke arsitektur di antara lapisan *input* dan *output* untuk memodelkan hubungan yang lebih kompleks dan bersifat nonlinier. Konsep tersebut menarik minat para peneliti dalam beberapa tahun terakhir untuk kinerja dari DL yang baik untuk menjadi solusi dalam banyak masalah dalam aplikasi analisis citra medis seperti *denoising* citra, segmentasi, dan klasifikasi (Litjens, 2017).

2.3 Convolutional Neural Network

Convolutional Neural Network (CNN) merupakan salah satu arsitektur *deep learning* untuk pengolahan citra, baik *semantic segmentation*, *instance segmentation*, *synthetic image generation*, dan *image classification*. CNN sebagai langkah awalnya dapat mengekstrak fitur yang bersifat *low-level* dari *input* mentah (*raw*) dan kemudian mengekstrak fitur yang bersifat global dan fitur *high-level* di

lapisan yang lebih dalam. Misalnya, CNN dapat belajar untuk melakukan deteksi fitur visual yang semakin kompleks dari gambar mentah, seperti deteksi tepi gambar, bentuk sederhana, maupun deteksi objek secara lengkap (Schirrmeister, 2017).

CNN dirancang untuk memproses data dalam bentuk *array*, misalnya gambar berwarna yang terdiri dari tiga buah *arrays* dua dimensi (2D) yang mengandung intensitas *pixel* dalam tiga saluran warna. Arsitektur CNN disusun sebagai serangkaian tahapan. Beberapa tahap awal terdiri dari dua jenis *layer*, yaitu *convolutional layer* dan *pooling layer*. Peran dari *convolutional layer* adalah untuk mendeteksi hubungan dari fitur-fitur pada lapisan sebelumnya, sedangkan peran dari *pooling layer* adalah untuk menggabungkan fitur serupa secara semantik menjadi satu (LeCun, 2015).

CNN memiliki catatan atau rekor yang mengesankan untuk aplikasi dalam analisis dan interpretasi gambar, termasuk citra untuk medis (Pratt, 2016). CNN dapat mengurangi tingkat kesalahan (*error*) dalam kasus identifikasi multilabel dengan menggunakan *dataset* ImageNet di mana 1.2 juta gambar harus diklasifikasikan ke dalam 1000 kelas yang berbeda. Tingkat kesalahan dari di atas 26% dapat menurun menjadi di bawah 4% (Krizhevsky, 2012). Secara garis besar CNN tidak jauh berbeda dengan *neural network* pada umumnya. CNN terdiri dari neuron yang memiliki *weight*, *bias* dan *activation function*. Adapun hal yang membedakan CNN dengan *neural network* yang lain adalah pada arsitekturnya yang dibedakan menjadi dua bagian yaitu *Feature Extraction Layer* dan *Fully-Connected Layer* (MLP). Sementara itu, kelebihan dari CNN adalah bahwa CNN sangat cocok digunakan untuk pembelajaran *end-to-end*, yaitu pembelajaran

menggunakan data mentah tanpa pemilihan fitur apriori dan memiliki performa yang baik untuk *dataset* berukuran besar (Szegedy, 2015).

2.3.1 Cara Kerja CNN

Peran CNN adalah untuk mereduksi gambar menjadi bentuk yang lebih mudah diproses, tanpa kehilangan fitur yang penting untuk mendapatkan prediksi yang baik. Hal tersebut menjadi penting ketika digunakan untuk merancang arsitektur yang tidak hanya bagus dalam mempelajari fitur tetapi juga dapat diskalakan ke kumpulan data besar. Hal yang membedakan CNN dengan *neural network* yang lain adalah pada arsitekturnya yang dibedakan menjadi dua bagian, yaitu *Feature Extraction Layer* dan *Fully-Connected Layer* (MLP). *Feature Extraction Layer* merupakan proses ekstraksi sebuah citra menjadi fitur yang berupa angka-angka yang merepresentasikan citra tersebut. *Feature Extraction Layer* terdiri dari dua bagian, yaitu *convolutional layer* dan *pooling layer*.

2.3.2 Convolutional Layer

Elemen yang terlibat dalam menjalankan operasi konvolusi pada bagian pertama di *convolutional layer* disebut sebagai *kernel* atau fitur. Tujuan dari operasi konvolusi adalah untuk melakukan ekstrak pada fitur *high-level* yang dimiliki oleh citra *input*, seperti garis tepi. Sebuah model CNN dapat memiliki lebih dari satu convolutional layer. Secara konvensional, lapisan *convolutional* yang pertama bertanggungjawab untuk melakukan *capturing* fitur yang bersifat *low-level*, seperti tepian gambar, warna, dan orientasi gradien. Dengan ditambahkannya lapisan, arsitektur juga dapat beradaptasi dengan fitur yang bersifat *high-level*,

sehingga menghasilkan *network* yang memiliki pemahaman yang baik tentang citra yang ada di dalam *dataset* (Liu, Shen dan Van Den Hengel, n.d.).

2.3.3 Pooling Layer

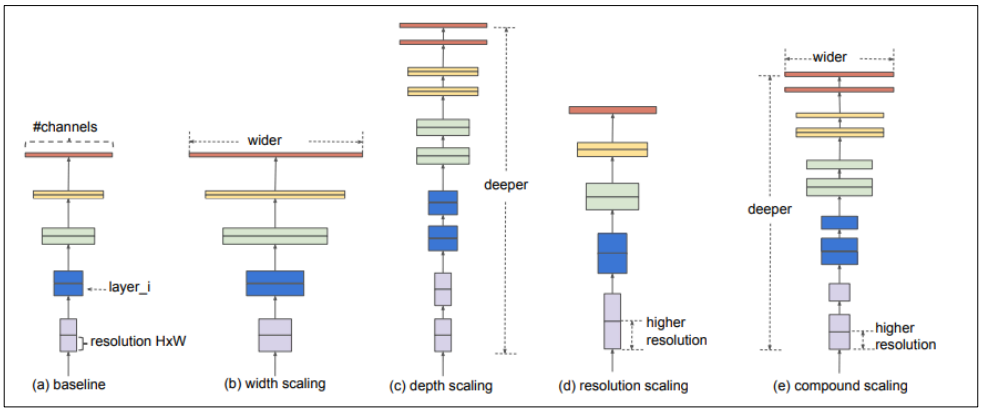
Memiliki kemiripan dengan *convolutional layer*, *pooling layer* berfungsi untuk mengurangi ukuran spasial fitur konvolusi. Hal tersebut bertujuan untuk mengurangi daya komputasi yang diperlukan untuk memproses data melalui reduksi dimensionalitas. Lebih lanjut, *pooling layer* berguna untuk mengekstraksi fitur dominan yang bersifat invarian rotasional dan posisional, sehingga dapat mempertahankan proses pelatihan model secara efektif. Tujuan penggunaan *pooling layer* adalah untuk mengurangi dimensi *feature map*, sehingga mempercepat komputasi karena parameter yang harus di-*update* semakin sedikit dan untuk mengatasi *overfitting*.

Terdapat dua jenis *pooling*, yaitu *Max Pooling* dan *Average Pooling*. *Max Pooling* mengembalikan nilai maksimum dari porsi gambar yang dicakup oleh *kernel*. Sementara itu, *Average Pooling* mengembalikan nilai rata-rata dari semua nilai di bagian gambar yang dicakup oleh *kernel*. *Convolutional layer* dan *pooling layer* membentuk lapisan ke-i dari CNN. Bergantung pada kompleksitas dalam gambar, jumlah lapisan tersebut dapat ditingkatkan untuk menangkap detail fitur yang bersifat *low-level* lebih jauh, akan tetapi hal tersebut berdampak terhadap biaya (*cost*) daya komputasi yang lebih besar (McKinley et al., 2018).

2.3.4 Fully Connected Layer

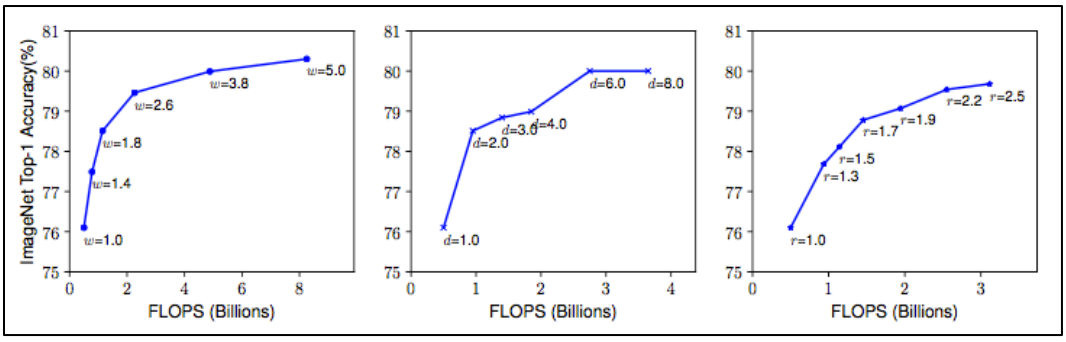
Fully connected layer merupakan MLP yang memiliki beberapa *hidden layer*, *activation function*, *output layer,* dan *loss function*. Setelah melalui *layer* yang sudah disebutkan di atas, model telah berhasil diaktifkan untuk memahami fitur yang ada pada citra. Selanjutnya fitur yang sudah dihasilkan akan menjadi *input* dari *neural network* untuk selanjutnya dilakukan proses klasifikasi. Dengan menambahkan *fully connected layer* merupakan cara yang umum digunakan untuk mempelajari kombinasi non-linier dari fitur *high-level* seperti yang ditunjukkan oleh *output* yang dihasilkan oleh *convolutional layer*

2.4 EfficientNet



Gambar 2.1 Model Scaling (Tan, 2019)

Dalam penggunaan CNN, *scaling* merupakan salah satu faktor penting untuk meningkatkan akurasi dan efisiensi dari model yang dihasilkan. Adapun tiga dimensi *scaling* pada CNN adalah *depth* , *width*, dan *resolution*. Penting untuk dilakukan penyeimbangan terhadap ketiga dimensi tersebut dan hal tersebut dapat dicapai dengan menggunakan rasio yang konstan untuk melakukan *scaling* pada tiap dimensi. EfficientNet dapat melakukan *scaling* secara seragam pada semua dimensi dengan menggunakan *compound coefficient* (Tan, 2019).



Gambar 2.2 Meningkatkan *Model Baseline* dengan Koefisien *width* (w), *depth* (d), dan *resolution* (r) yang berbeda (Tan, 2019)

2.4.1 Depth

Melakukan *scaling* pada *depth* merupakan cara yang paling sering dilakukan oleh kebanyakan CNN. *Depth* mengacu pada jumlah layer atau kedalaman dari suatu *network* (Szegedy, 2015). CNN dengan jumlah *layer* yang banyak dan lebih dalam dapat menangkap fitur yang lebih banyak dan lebih kompleks, serta dapat melakukan generalisasi tugas-tugas baru dengan baik. Akan tetapi, *network* yang lebih dalam juga lebih sulit untuk dilatih karena gradien yang semakin menghilang (Zagoruyko & Komodakis, 2016). Meskipun beberapa teknik, seperti normalisasi *batch* meringankan masalah pelatihan, perolehan akurasi dari *network* yang sangat dalam menjadi berkurang, seperti misalnya ResNet-1000 memiliki akurasi yang sama seperti ResNet-101 meskipun memiliki lebih banyak *layer* (Ioffe & Szegedy, 2015). Pada Gambar 2.2 (tengah) merupakan grafik untuk nilai akurasi yang dihasilkan oleh model dengan penambahan nilai *depth.* Dapat dilihat bahwa untuk nilai depth yang semakin besar, penambahan nilai akurasi menjadi tidak terlalu signifikan (penambahan nilai akurasi dari kenaikan nilai *depth* 4.0 menjadi 6.0 tidak sebesar penambahan nilai akurasi dari kenaikan nilai *depth* 1.0 menjadi 2.0)

2.4.2 Width

Scaling dengan menggunakan *width* banyak dilakukan untuk model yang memiliki ukuran kecil. *Network* yang lebih lebar cenderung mampu menangkap lebih banyak fitur yang berukuran kecil dan lebih mudah untuk dilatih. Namun, jaringan yang sangat luas tetapi tidak dalam cenderung mengalami kesulitan dalam menangkap fitur pada tingkat yang lebih tinggi (Zagoruyko & Komodakis, 2016). Pada Gambar 2.2 (kiri) dapat dilihat bahwa kenaikan akurasi menjadi tidak terlalu signifikan ketika nilai *width* dari model semakin besar.

2.4.3 Resolution

Resolution mengacu pada resolusi dari gambar *input*. Dengan gambar *input* yang memiliki resolusi lebih tinggi, CNN lebih berpotensi untuk menangkap pola-pola yang lebih halus. Mulai dari 224x224 di CNN awal, sementara pada saat ini CNN cenderung menggunakan 299x299 atau 331x331 untuk mendapatkan akurasi yang lebih baik (Tan, 2019). Resolusi yang lebih tinggi, seperti 600x600, juga banyak digunakan oleh CNN untuk mendeteksi objek (He, 2016). Dengan demikian, memperbesar *depth*, *width*, dan *resolution* dapat meningkatkan akurasi. Akan tetapi peningkatan akurasi akan menurun dengan ukuran model yang semakin besar (Tan, 2019).

2.4.4 Persamaan

Layers atau lapisan dari *Convolutional Network* (ConvNet) sering dibagi menjadi ke dalam beberapa *stages* dan semua lapisan di setiap *stage* mempunyai arsitektur yang sama, seperti ResNet yang mempunyai lima *stages* dan setiap stage

mempunyai tipe yang sama, kecuali pada lapisan pertamanya yang melakukan *down-sampling.* Sebuah ConvNet dapat direpresentasikan sebagai $\mathcal{N}$, di mana $\mathcal{N}$ memiliki 1 sampai dengan s *stage* dan setiap *stage* terdiri dari L_i *layer* dengan ukuran kernel sebesar $\langle H_i, W_i, C_i \rangle$, di mana H_i dan W_i merupakan *spatial dimension* (tinggi dan lebar) serta C_i merupakan *channel dimension* Adapun ConvNet dapat didefinisikan sebagai berikut.

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i} \left(X_{\langle H_i, W_i, C_i \rangle}\right) \tag{2.1}$$

Pada Rumus 2.1, $\mathcal{F}_i^{L_i}$ menggambarkan bahwa lapisan $\mathcal{F}_i$ diulang sebanyak L_i pada *stage* i. Sementara itu, $\langle H_i, W_i, C_i \rangle$ menunjukkan ukuran *input* dari *tensor* X pada layer i. Tidak seperti ConvNet pada umumnya yang fokus dalam menemukan arsitektur $\mathcal{F}_i$, pada *model scaling* nilai dari L_i, H_i, W_i, dan C_i diperbesar tanpa mengubah $\mathcal{F}_i$ yang sudah ditentukan pada *baseline* dari *network*. Untuk mengurangi kemungkinan nilai yang banyak dari L_i, H_i, W_i, dan C_i, semua *layers* diskalakan secara seragam dengan rasio yang bernilai konstan. Tujuan yang ingin dicapai oleh EfficientNet adalah memaksimalkan akurasi dari model yang dihasilkan dengan adanya keterbatasan *resource*, seperti ukuran memori dan jumlah *Floating Point Operations Per Seconds* (FLOPS) dari GPU. Hal tersebut dapat dirumuskan pada Rumus 2.2 sebagai berikut.

$$\begin{gathered} \max_{d,w,r} Accuracy(\mathcal{N}(d,w,r)) \\ s.t.\ \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d.\hat{L}_i} \left(X_{\langle r.H_i, r.W_i, w.C_i \rangle}\right) \\ \text{Memory } (\mathcal{N}) \leq \text{target\_memory} \\ \text{FLOPS } (\mathcal{N}) \leq \text{target\_flops} \end{gathered} \tag{2.2}$$

Pada Rumus 2.2, d, w, r merupakan koefisien yang digunakan untuk melakukan *scaling* pada *depth*, *width*, dan *resolution* dari *network*. $\hat{\mathcal{F}}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ merupakan parameter yang nilainya sudah ditentukan pada *baseline* dari *network* seperti contoh yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 *Baseline* dari *Network* pada EfficientNet-B0

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i$ x $\hat{W}_i$	#Channels $\hat{C}_i$	#Layers L_i
1	Conv3x3	224 x 224	32	1
2	MBConv1, k3x3	112 x 112	16	1
3	MBConv6, k3x3	112 x 112	24	2
4	MBConv6, k5x5	56 x 56	40	2
5	MBConv6, k3x3	28 x 28	80	3
6	MBConv6, k5x5	14 x 14	112	3
7	MBConv6, k5x5	14 x 14	192	4
8	MBConv6, k3x3	7 x 7	320	1
9	Conv1x1 & Pooling & FC	7 x7	1280	1

EfficientNet menggunakan metode *compound scaling* dengan menggunakan *compound coefficient* ϕ untuk melakukan *scaling* pada *width, depth, dan resolution* dengan seragam yang memenuhi persamaan pada Rumus 2.3 berikut.

$$\begin{aligned} &\text{depth} : d = \alpha^{\phi} \\ &\text{width} : w = \beta^{\phi} \\ &\text{resolution} : r = \gamma^{\phi} \\ &s.t.\ \alpha.\beta^2.\gamma^2 \approx 2 \\ &\alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{2.3}$$

$\alpha, \beta,$ dan γ merupakan konstanta yang nilainya dapat ditentukan dengan menggunakan Grid Search. Sementara itu, ϕ merupakan koefisien yang nilainya ditentukan oleh pengguna yang digunakan untuk mengontrol besar *resource* yang tersedia untuk melakukan *scaling* pada model. $\alpha, \beta,$ dan γ menspesifikasikan

bagaimana menentukan *resource* yang tersedia ke *width, depth*, dan *resolution* dari network. Melakukan *scaling* pada model dengan menggunakan persamaan pada Rumus 2.3 dapat meningkatkan total dari FLOPS sebesar $(\alpha.\beta^2.\gamma^2)^\phi$ karena FLOPS berbanding lurus dengan d, w^2, dan r^2. Dengan menggandakan nilai dari *depth* akan turut menggandakan nilai dari FLOPS, sementara itu menggandakan nilai dari *width* dan *resolution* akan menghasilkan FLOPS dengan nilai empat kali lebih besar. Pembatasan $\alpha.\beta^2.\gamma^2 \approx 2$ bertujuan supaya untuk setiap nilai ϕ yang baru, total dari FLOPS diperkirakan bertambah hingga 2^ϕ (Tan, 2019).

2.5 Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) dikembangkan oleh Chen dan Guestrin pada tahun 2016 dan merupakan varian dari algoritma *Tree Gradient Boosting*. Algoritma ini merupakan interpretasi dari metode *Newton Boosting* karena kesamaan dalam konsep algoritmanya. XGBoost merupakan kombinasi antara *Gradient Descent* dan *Boosting* yang disebut *Gradient Boosting Machine* (GBM). *Boosting* adalah algoritma *Ensemble Learning* yang memberikan bobot yang berbeda untuk distribusi data pelatihan pada setiap iterasi. Dengan *gradient boosting* sebagai model dasarnya, XGBoost menggabungkan model dengan pembelajaran dasar yang lemah menjadi lebih kuat dengan menggunakan metode pembelajaran secara berulang.

Pada *gradient boosting*, di setiap iterasinya, *residual* (perbedaan antara perkiraan yang diperoleh pada saat ini dengan vektor target yang sudah diketahui) akan digunakan untuk memperbaiki *predictor* sebelumnya. Dengan hal tersebut, *loss function* yang sudah ditentukan dapat dioptimalkan. Sebagai perbaikan dari

gradient boosting tersebut, regularisasi ditambahkan *pada loss function* guna menetapkan *objective function* (cara untuk mengukur seberapa baik model menghasilkan keluaran yang tepat terhadap target) untuk mengukur performa model yang dihasilkan oleh XGBoost (Zhang, et al., 2018)

XGBoost memiliki parameter tambahan dibandingkan dengan *Gradient Boosting* lain. Parameter tersebut berupa nilai pinalti pada tiap *tree*. Nilai pinalti tersebut memberikan pengaruh terhadap nilai pada struktur pohon baru dan memberikan bobot dari pencabangan yang bertujuan untuk mengurangi variasi pada setiap pohon. Faktor terpenting di balik kesuksesan XGBoost adalah skalabilitasnya (*scalability*) pada semua skenario. Sistem berjalan lebih dari sepuluh kali lebih cepat dibandingkan dengan solusi populer yang ada pada satu mesin dan menskalakan hingga miliaran data dalam pengaturan memori yang terbatas atau terdistribusi. Skalabilitas XGBoost disebabkan oleh beberapa sistem penting dan pengoptimalan algoritmik. Komputasi paralel dan terdistribusi membuat pembelajaran lebih cepat yang memungkinkan eksplorasi model lebih cepat. Selain itu, XGBoost juga mengeksploitasi komputasi *out-of-core* dan memungkinkan seorang *data scientist* untuk dapat memproses ratusan juta data pada desktop (Chen & Guestrin, 2016).

2.6 Confusion Matrix

Confusion matrix merupakan sebuah konsep dari *machine learning* yang mengandung informasi mengenai nilai aktual dan hasil klasifikasi yang dilakukan oleh sebuah model klasifikasi. Informasi yang terdapat pada *confusion matrix* tersebut dapat membantu dalam menyempurnakan klasifikasi atau perkiraan yang

diturunkan dari klasifikasi tersebut. Sebuah *confusion matrix* memiliki dua dimensi, di mana indeks pertama merupakan indeks dari kelas aktual atau kelas sebenarnya dari sebuah objek dan yang kedua merupakan indeks dari kelas hasil prediksi yang dilakukan oleh model klasifikasi. Performa dari model klasifikasi yang dihasilkan dievaluasi dengan menggunakan data yang ada pada matriks tersebut (Deng, 2016). Representasi dari *confusion matrix* dapat dilihat pada Tabel 2.2 berikut.

Tabel 2.2 Representasi *Confusion Matrix*

Aktual	Prediksi	
	Negatif	Positif
Negatif	TN	FP
Positif	FN	TP

Berdasarkan Tabel 2.2, TN atau *True Negative* merupakan nilai dari prediksi yang dilakukan dengan benar bahwa suatu objek bernilai negatif, FP atau *False Positive* merupakan nilai dari prediksi yang dilakukan dengan salah bahwa suatu objek bernilai positif, FN atau *False Negative* merupakan nilai dari prediksi yang dilakukan dengan salah bahwa suatu objek bernilai negatif, dan TP atau *True Positive* merupakan nilai dari prediksi yang dilakukan dengan benar bahwa suatu objek bernilai positif. Beberapa *metrics* yang dapat dihitung dengan menggunakan nilai *a, b, c,* dan *d* pada *confusion matrix* adalah *accuracy, precision, recall, specificity*, dan F1 *score* (Sokolova dan Lapalme, 2009).

2.6.1 Accuracy

Accuracy atau akurasi merupakan perbandingan dari prediksi yang dengan benar dilakukan oleh model dengan total prediksi yang dilakukan (Deng, 2016). Untuk memperoleh nilai akurasi dapat menggunakan persamaan pada Rumus 2.4 berikut.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (2.4)$$

2.6.2 Precision

Precision merupakan perbandingan dari kelas bernilai positif yang dengan benar diklasifikasikan oleh model dengan keseluruhan kelas yang diprediksi bernilai positif. Secara sederhana, *precision* dapat diartikan sebagai proposi kelas dengan nilai positif yang berhasil diklasifikasikan dengan benar (Sokolova dan Lapalme, 2009). Untuk memperoleh nilai dari *precision* dapat menggunakan persamaan pada Rumus 2.5 berikut.

$$Precision = \frac{TP}{TP + FP} \quad (2.5)$$

2.6.3 Recall

Recall atau dapat disebut sebagai *sensitivity* dari model klasifikasi atau *True Positive Rate* (TPR) merupakan kelas yang dengan benar diklasifikasikan ke dalam kelas bernilai positif dengan keseluruhan kelas yang memang bernilai positif. *Recall* dapat diartikan sebagai proposi dari kelas yang memiliki nilai aktual positif yang diklasifikasikan dengan benar (Sokolova dan Lapalme, 2009). Nilai dari *recall* atau TPR dapat diperoleh melalui persamaan pada Rumus 2.6 berikut.

$$Recall = \frac{TP}{TP + FN} \quad (2.6)$$

2.6.4 Specificity

Specificity atau *True Negative Rate* (TNR) merupakan kebalikan dari *recall* atau TPR. *Specificity* mengukur perbandingan antara kelas yang dengan benar

diklasifikasikan ke dalam kelas bernilai negatif dengan keseluruhan kelas yang memang bernilai negatif. Dengan diperolehnya nilai *specificity*, nilai dari *False Positive Rate* (FPR) juga dapat diperoleh. FPR merupakan proporsi dari kelas bernilai negatif yang diklasifikasikan sebagai bernilai positif (Deng, 2016). Adapun untuk memperoleh nilai *specificity* dan FPR dari suatu model klasifikasi dapat dilihat pada Rumus 2.7 dan Rumus 2.8 sebagai berikut.

$$Specificity = \frac{TN}{TN + FP} \tag{2.7}$$

$$\begin{aligned} FPR &= 1 - Specificity \\ &= \frac{FP}{FP + TN} \end{aligned} \tag{2.8}$$

2.6.5 F1 Score

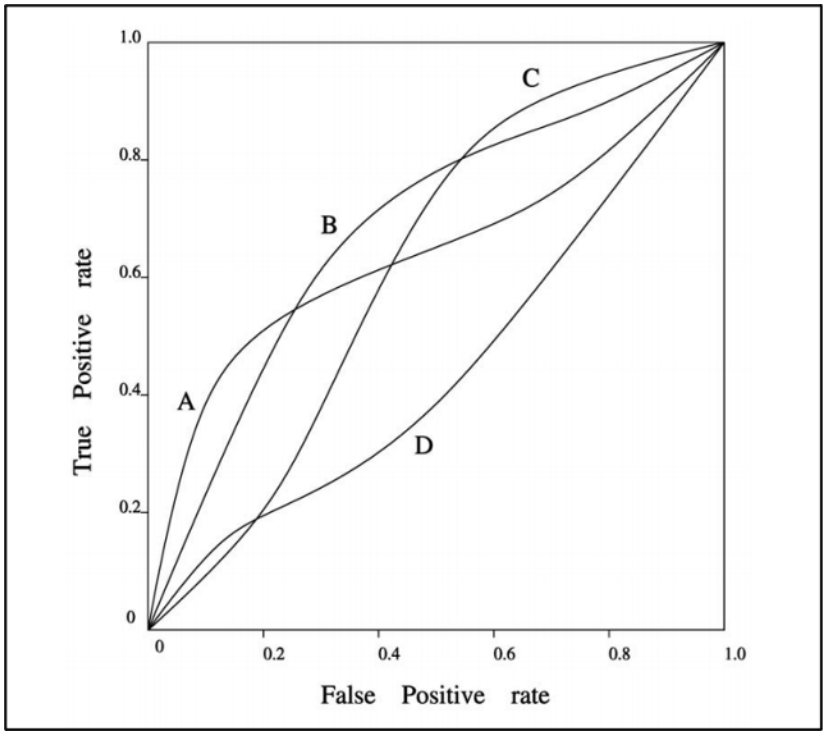
Dengan diperolehnya nilai-nilai yang sudah dijelaskan sebelumnya (*accuracy, precision, recall*, dan *specificity*) dapat digunakan untuk memperoleh nilai dari F1 Score atau F-Score. F1 Score merupakan rata-rata (mean) dari *precision* dan *recall* serta memiliki nilai maksimum satu (1) dan nilai minimumnya nol (0). Ketepatan dan kapabilitas dari model klasifikasi yang dihasilkan dapat diukur dengan menggunakan F1 Score (Goutte dan Gaussier, 2015). Rumus 2.9 berikut digunakan untuk memperoleh nilai dari F1 Score.

$$F1\ Score = \frac{2TP}{2TP + FP + FN} \tag{2.9}$$

2.7 Area Under the Curve of Receiver Operating Characteristic

Tujuan dari *classification learning algorithm* adalah untuk menghasilkan *classifier* dari sekumpulan data *training* yang telah diberi label supaya model tersebut dapat melakukan prediksi terhadap data *testing* yang tidak diketahui. Performa dari *classifier* tersebut diukur berdasarkan akurasi dari prediksi terhadap data *testing*. Salah satu cara untuk mengukur performa dari sebuah *classifier* adalah dengan menggunakan *Area Under the Curve of Receiver Operating Characteristic* (ROC AUC). ROC merupakan sebuah kurva probabilitas dengan AUC sebagai area di bawah kurva tersebut. Nilai maksimum dari AUC adalah satu (1) yang mengindikasikan bahwa model klasifikasi tersebut memiliki nilai yang sempurna atau model klasifikasi tersebut seratus persen (100%) sensitif dan seratus persen (100%) spesifik (Fan, Upadhye dan Worster, 2006).

ROC AUC dapat memberikan informasi mengenai kemampuan dari *classifier* untuk membedakan antar kelas dan pertama kali digunakan dalam teori deteksi sinyal untuk mewakili *tradeoff* antara *hit rate* dengan *false alarm rate.* ROC AUC kemudian banyak digunakan dalam diagnosis medis sejak tahun 1970-an. Spackman merupakan salah satu peneliti pertama yang menggunakan ROC AUC untuk membandingkan dan mengevaluasi algoritma *machine learning* (Jin Huang dan Ling, 2005). Pada Gambar 2.4 berikut merupakan contoh dari kurva ROC AUC. Dapat dilihat pada Gambar 2.4 bahwa kurva ROC AUC digambarkan dengan *True Positive Rate* (TPR) terhadap *False Positive Rate* (FPR atau [1-*specificity*]) di mana TPR berada pada sumbu y dan FPR pada sumbu x.



Gambar 2.3 Contoh Kurva ROC AUC (Jin Huang dan Ling, 2015)

Kurva ROC AUC membantu dalam memberikan ilustrasi secara grafis mengenai *tradeoff* di antara *sensitivity* dan *specificity* dari suatu model klasifikasi. Apabila puncak dari suatu kurva semakin mendekati ujung kiri atas, di mana nilai dari *sensitivity* dan *specificity* merupakan yang tertinggi, maka model klasifikasi tersebut memiliki kemampuan untuk melakukan klasifikasi antar kelas yang semakin baik atau nilai TPR semakin besar dan sebaliknya nilai dari FPR semakin kecil (Fan, Upadhye dan Worster, 2006). Berdasarkan Gambar 2.3, kurva A dan kurva B memiliki nilai AUC yang lebih baik apabila dibandingkan dengan kurva D karena posisi dari kurva A dan B tersebut terletak di atas kurva D dan lebih mendekati ujung kiri atas apabila dibandingkan dengan kurva D (Jin Huang dan Ling, 2015).